dBFT implementation

Consensus Project

CS21B077 Shruthi Kannappan
CS20B029 Tej Pavan

May 5, 2024

1 Introduction

Consensus algorithms are crucial for maintaining security, reliability, and efficiency in blockchain networks. Delegated Byzantine Fault Tolerance (dBFT) is one such mechanism that ensures network integrity by reaching consensus even when some nodes are faulty or malicious. This project focuses on implementing a dBFT-based consensus system, examining its architecture, operational processes, and resistance to Byzantine faults.

The dBFT algorithm is significantly more energy-efficient compared to other consensus algorithms. dBFT relies on a group of consensus nodes (called "validators & delegates") to validate transactions and achieve consensus, which is a more energy-efficient process.

2 Overview of Consensus Algorithm

Nodes elect a subset of nodes as "validators" and "delegates" who are like their representatives who participate in various verification stages of dBFT on their behalf.
Some Terminologies:
Consensus Node : Any node participating in the consensus algorithm.
Speaker: A consensus node whose nodeID is the same as the view number (Leader node), the one who proposes the sequence number for the transactions it received.
n: Number of consensus nodes.
f: The maximum number of Byzantine nodes.
Validator: A consensus node who is elected by the consensus nodes. It verifies prepare messages and broadcasts a commit message on receiving $2f + 1$ prepare messages.
Delegate: A validator who verifies the pre-prepare message and broadcasts a prepare message.
Phase : A set of n view changes
Each view proceeds as follows:

1. The Speaker processes and assigns a sequence number to the message sent by the client and broadcasts Preprepare message to delegates.

2. When Delegate receives the Preprepare request it waits for a total of $2f+1$ preprepare requests and broadcasts Prepare message to Validators.

3. When Validator receives the Prepare request it waits for a total of $2f+1$ prepare requests and broadcasts commit message to consensus nodes

4. When consensus node receives the commit message it waits for a total of $2f+1$ commit messages and it commits the transaction.

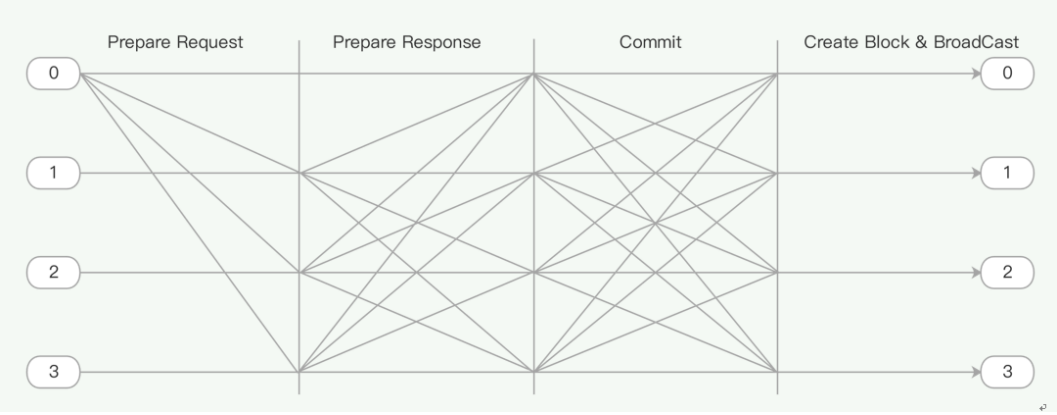

Figure 1: Phases of the algorithm

Each Delegate maintains a timeout period foe each view. When the timer runs out it proposes a view change to all other nodes. When a consensus node has atleast $2f+1$ view change requests, then it changes it's view.

The view changes in a phase are numbered from 0 to n-1.

After every n view changes election is performed to decide the Validators and Delegates.

3 Implementation Design & Details

This setup contains 7 consensus nodes, 5 Validators and 4 Delegates.

Each node does the following:

- Maintains a list of Validators and Delegates.
- If it is a Delegate it maintains timer for the current view.
- Reset view number to 0 after every n view changes.
- Adds committed-local messages to it's ledger.

Election proceeds as follows:

1. Each node generates a random number between 0 to 100 and broadcast it as it's vote to every other consensus node and keeps track of votes it received.

2. Each node waits for timeout period for all the consensus nodes to cast their vote.

3. After the timeout period each node sorts the votes according to the value associated with it and decides the Delegates as best 4 candidates and Validators as best 5 candidates.

Note :- Assuming every node behaves well during election.(i.e, election implemented here is not Byzantine fault tolerant). It is however clean crash tolerant.

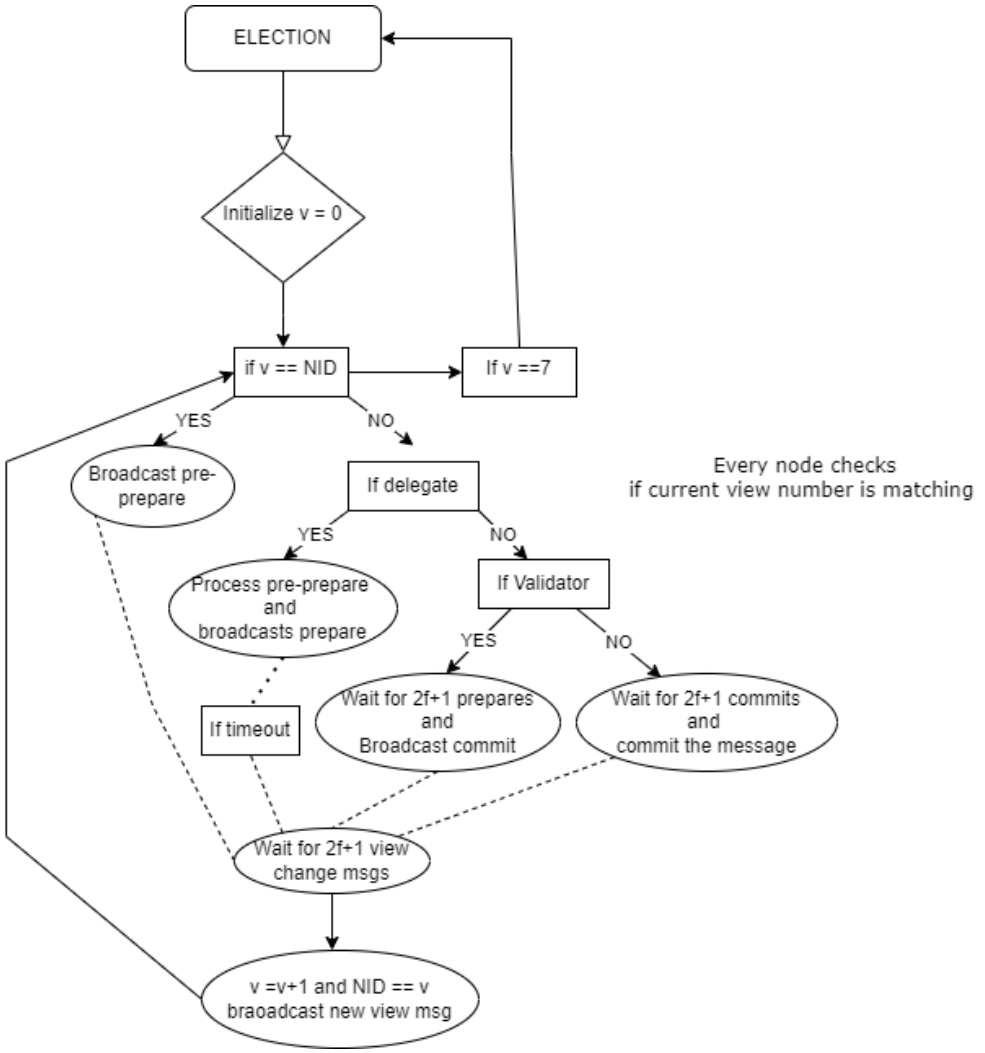

Figure 2: flow of the algorithm

4 Fault Tolerance in Implementation

Our consensus implementation works for atmost 1 Byzantine node failure (or) crash failure. Our implementation works well even if the Byzantine node is speaker or delegate or validator, through view change mechanism.

5 Include Testing

We have tested for various crash failures in vms, and the code is running as intended. The code works successfully if there is one Byzantine node.

6 Conclusion

Since nodes elect a subset of nodes as "validators" and "delegates" who are like their representatives and do the verification of the proposed transactions by a speaker on their behalf. This reduces the overall compute of the system. This is much more efficient compared to PBFT where every node needs to go through every stage of the algorithm. Moreover, the current implementation brings in a set of validators and delegates in each phase introducing randomization.

A possible could be implementing a Byzantine tolerant election protocol where the consensus nodes elect their candidates based on their behaviour.